8. Control the number of ports opened by the operating system with

a) Semaphore b) Monitors.

//semaphore

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t mutex;

void\* thread(void\* arg)

{

//wait

sem\_wait(&mutex);

printf("\n Enter thread\n");

//critical section

sleep(4);

//signal

printf("\n Exit thread\n");

sem\_post(&mutex);

}

int main()

{

sem\_init(&mutex,0,1);

pthread\_t t1,t2;

pthread\_create(&t1,NULL,thread,NULL);

sleep(2);

pthread\_create(&t2,NULL,thread,NULL);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

sem\_destroy(&mutex);

return 0;

}

Output: $semaphore.c - o semaphore

semaphore enter

Enter Thread

Exit Thread

Enter Thread

Exit Thread

//b.)Monitors:

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <semaphore.h>

// Define the data we need to create the monitor

struct monitor\_DataType {

sem\_t OKtoRead;

sem\_t OKtoWrite;

int readerCount;

int isBusyWriting;

// The read-queue

int readRequested;

};

struct monitor\_DataType monitor\_data;

// Function that will block until write can start

void monitor\_StartWrite() {

if(monitor\_data.isBusyWriting || monitor\_data.readerCount != 0){

sem\_wait(&(monitor\_data.OKtoWrite));

}

monitor\_data.isBusyWriting++; // Using 1 as true

}

// Function to signal reading is complete

void monitor\_EndWrite() {

monitor\_data.isBusyWriting--;

if(monitor\_data.readRequested){

sem\_post(&(monitor\_data.OKtoRead));

} else {

sem\_post(&(monitor\_data.OKtoWrite));

}

}

// Function that will block until read can start

void monitor\_StartRead() {

if(monitor\_data.isBusyWriting){

monitor\_data.readRequested++;

sem\_wait(&(monitor\_data.OKtoRead));

monitor\_data.readRequested--;

}

monitor\_data.readerCount++;

sem\_post(&(monitor\_data.OKtoRead));

}

// Function to signal reading is complete

void monitor\_EndRead() {

monitor\_data.readerCount--;

if(monitor\_data.readerCount == 0){

sem\_post(&(monitor\_data.OKtoWrite));

}

}

// intialize the monitor

// return's 0 on success, just like sem\_init()

int monitor\_Initialized(){

int returnValue = 1;

// Initialize the structure

monitor\_data.readerCount = 0;

monitor\_data.isBusyWriting = 0;

monitor\_data.readRequested = 0;

// initialize the semaphores

if(sem\_init(&(monitor\_data.OKtoWrite), 0, 1) == 0 &&

sem\_init(&(monitor\_data.OKtoRead), 0, 1) == 0){

returnValue = 0;

} else {

printf("Unable to initialize semaphores\n");

}

return returnValue;

}

// Destroys the semphores.

void monitor\_Destroy(){

sem\_destroy(&(monitor\_data.OKtoWrite));

sem\_destroy(&(monitor\_data.OKtoRead));

}

int main() {

/\*\* For testing only

--------------------------------

if(monitor\_Initialized() == 0){

printf("Initialized\n");

monitor\_StartWrite();

printf("Writing stuffs...\n");

monitor\_EndWrite();

monitor\_StartRead();

printf("Reading stuffs...\n");

monitor\_EndRead();

monitor\_Destroy();

}

\*/

return 0;

}

Output:

Result:Successfully executed semaphore and monitor.